UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/666,032	09/18/2003	Eric Lawrence Barsness	ROC920030264US1	7940

46296 7590 05/25/2010
MARTIN & ASSOCIATES, LLC
P.O. BOX 548
CARTHAGE, MO 64836-0548

EXAMINER	
MITCHELL, JASON D	

ART UNIT	PAPER NUMBER
2193	

MAIL DATE	DELIVERY MODE
05/25/2010	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

Ex parte ERIC LAWRENCE BARSNESS,
GREGORY CHRISTOPHER PLACE,
and JOHN MATTHEW SANTOSUOSSO

Appeal 2009-002971
Application 10/666,032[1]
Technology Center 2100

Decided: May 25, 2010

Before LEE E. BARRETT, JEAN R. HOMERE, and THU A. DANG, *Administrative Patent Judges*.

BARRETT, *Administrative Patent Judge*.

DECISION ON APPEAL

This is a decision on appeal under 35 U.S.C. § 134(a) from the final rejection of claims 8, 9, 17, 18, 28, and 29. Claims 1-7, 10-16, 19-27, and 30-32 are canceled. We have jurisdiction pursuant to 35 U.S.C. § 6(b).

We affirm.

---

[1] Filed September 18, 2003, titled "Inter-Job Breakpoint Apparatus and Method." The real party in interest is International Business Machines Corporation.

STATEMENT OF THE CASE

The invention

The invention relates to debugging computer programs when there are many jobs running at the same time. The invention provides inter-job breakpoints by defining a condition for a first job and an action for a second job. When the condition in the first job is satisfied, the action in the second job is performed. The condition in the first job can be the start of execution of a specified portion of code in the first job, the end of execution of a specified portion of code in the first job, or the satisfying of some other condition in the first job. The action in the second job can be the halting of the second job or the enabling of a breakpoint in the second job. Inter-job breakpoints enhance the utility of a debugger by performing an action in one job based on a detected condition in a different job. Spec. 2.

Illustrative claim

Claim 8 is reproduced below for illustration:

> 8. An apparatus comprising:
>
> at least one processor;
>
> a memory coupled to the at least one processor;
>
> a first job residing in the memory and executed by the at least one processor;
>
> a second job residing in the memory and executed by the at least one processor;

> an inter-job breakpoint mechanism that detects at least one condition in the first job and, in response thereto, modifies a program variable in the second job.

The references

Diec	6,083,281	Jul. 4, 2000
Heinen, Jr.	Re. 36,852	Sept. 5, 2000

Dieter Haban and Wolfgang Weigel, *Global Events and Global Breakpoints in Distributed Systems*, Proc. of Twenty-First Annual Hawaii Int'l Conf. on Software Track, 166-175 (1988) ("Haban").

Microsoft Computer Dictionary 547 (5th ed. Microsoft Press 2002).

"Process (computing)," http://en.www.wikipedia.org/wiki/Process (computing) (last visited Jan. 5, 2009).

The rejections

Claims 8, 17, and 28 stand rejected under 35 U.S.C. § 103(a) as unpatentable over Haban and Heinen.

Claims 9, 18, and 29 stand rejected under 35 U.S.C. § 103(a) as unpatentable over Haban and Heinen, further in view of Diec.

DISCUSSION

Claims 8, 17, and 28

Contentions

The Examiner finds that Haban teaches jobs residing in memories of different processors and that the global breakpoint mechanism in Haban is

"an inter-job breakpoint mechanism that detects at least one condition in the first job," as recited in claim 8 and, in response thereto, sends a message to a second job which causes an action to be performed. Final Office Action (FOA) 5. The Examiner finds that Haban does not describe that the message "modifies a program variable in the second job." The Examiner finds that Heinen teaches that a message modifies a program variable in a second job at column 7, lines 30-32, because it teaches that a message requests that data be deposited in memory. FOA 5. The Examiner concludes that it would have been obvious to combine the teachings of Haban and Heinen "to provide means for debugging distributed processes." FOA 6.

Issue

Does the combination of Haban and Heinen teach or suggest "an inter-job breakpoint mechanism that detects at least one condition in the first job and, in response thereto, modifies a program variable in the second job," as recited in claim 8?

Claims 17 and 28 contain corresponding limitations and, thus, the rejection of claims 8, 17, and 28 stands or fall with claim 8.

Principles of law

"[T]he test [for obviousness] is what the combined teachings of the references would have suggested to those of ordinary skill in the art." *In re Keller*, 642 F.2d 413, 425 (CCPA 1981). A rejection under 35 U.S.C. § 103(a) is based on the following factual determinations: (1) the scope and content of the prior art; (2) the level of ordinary skill in the art; (3) the

differences between the claimed invention and the prior art; and (4) any objective indicia of non-obviousness. *KSR Int'l Co. v. Teleflex Inc.*, 550 U.S. 398, 399 (2007) (citing *Graham v. John Deere Co.*, 383 U.S. 1, 17 (1966)). All claim limitations must be taught or suggested.

Findings of fact

Specification

The Specification describes "program variables" as follows:

> Computer programs typically include one or more program variables that contain data of interest. These variables are typically represented by text labels in high-level and intermediate code computer programs. The concept of program variables is well known in the art.

Spec. 5, ll. 5-8.

The Specification describes a "condition" and "action" as follows:

> The inter-job breakpoint mechanism 126 allows setting a breakpoint that specifies a condition 127 in a first job and a corresponding action 128 on a second job. When the condition 127 is satisfied, the corresponding action 128 is performed. . . . Examples of actions 128 within the scope of the preferred embodiments include the halting of the second job, the enabling of a breakpoint in the second job, the modifying of a variable or other property on the second job, and the outputting of a debug message to the second job's output.

Spec. 7, l. 24 to Spec. 8, l. 8.

Haban

Haban relates to the problems of setting global breakpoints in distributed systems, how to detect those breakpoints, and how to halt the system in a consistent state. Abstract.

Global breakpoints are used in debugging distributed programs. P. 166, left col. A distributed program is a program consisting of multiple processes running on multiple processors. There are many problems for distributed programs. "Ideally, users should be able to specify assertions such as, 'halt the system when variable x of process b minus variable y of process c gets lower than zero." P. 166, left col. "The processes do not share memory but communicate via message passing." P. 166, left col.

"Global breakpoints are associated with the occurrence of global events. If a global event occurs and a breakpoint is associated with that global event, the distributed system is halted." P. 166, right col.

"*Primitive events* are the simplest form of global events." P. 167, left col.

User processes run on separate nodes and each node has its own debugging process or local debugger. P. 172, left col. The local debuggers propagate primitive events to other local debuggers. P. 172, right col.

In the implementation shown in Figure 17, a Test and Measurement Processor (TMP) network connects the nodes to a central test station (CTS). Haban describes:

> All the debuggers' communication is transmitted over the TMP network. Therefore, the basic communication between the application processes is not disturbed by the debugger's communication. The

> local debuggers evaluate the breakpoint tree and exchange information among each other without involving the central test station. The CTS is only informed when the global event is satisfied and the action associated with this satisfaction is performed (usually the halting of the distributed system).

P. 173, right col.

Haban further describes:

> We make a distinction between standard and optional *primitive events*. . . . Optional events are part of higher level programs that are generated by software development tools. . . . Optional events serve in additional application specific test and debug purposes such as setting global breakpoints. Each of these *primitive events* are represented by 1 byte data and each additional parameter by 4 bytes . . . . With the aid of a database gathered during unit compilation, the event processing software is able to interpret the incoming events and their parameters and present then in an application-oriented manner.

Paragraph bridging pp. 173-74.

Heinen

Heinen relates, as shown in Figure 1, to a debugger for debugging, from a central location (e.g., a user terminal 13), jobs or processes running on one or more remote units 11 connected to the user terminal 13 via a communication network 15. As shown in Figure 2, the user terminal 13 includes a debugger 21 that receives and interprets debug commands produced by a keyboard and display console 19. The debug commands fall in one of three categories--debug commands directed to the user terminal (USER TERMINAL CONTROL commands); debug commands directed to

a particular remote unit (REMOTE UNIT CONTROL commands); and, debug commands directed to a specific job or process of multiple jobs or processes running on a particular remote unit (LOCAL JOB/PROCESS commands). *See* Abstract. The rejection relies on a LOCAL JOB/PROCESS command.

The LOCAL JOB/PROCESS commands are transmitted to the remote units 11 via the communication network 15. Each of the remote units 11 include a remote unit debugger 23 and a local job/process debugger 25 for each job or process subject to debug control as shown in Figure 2. The local process debuggers 25 receive and carry out the LOCAL JOB/PROCESS commands. *See* Abstract.

The Examiner relies on the following statement in Heinen:

> Examples of LOCAL JOB/PROCESSES user commands are: . . . DEPOSIT -- a message requesting that data forming part of the message be deposited in the memory of the specified job or process.

Col. 7, ll. 27-32.

Analysis

1.

Haban teaches a debugger operating on a distributed system. Each process at a node is considered a "job" executed on a "processor." Consider the simple case of detecting a "primitive event," which corresponds to detecting a "condition in the first job," and the "global event" consists of this one primitive event. Consider the simple case of two nodes. In response to the detection of a primitive event, a message indicating the primitive event is

sent to the other local debugger over a network and includes a timestamp. P. 172, right col. The local debuggers are strongly connected to the processes, p. 172, so the debugger to which a message is sent is a "second job." The only issue is whether the message "modifies a program variable in the second job." Appellants provide no special details of how this is done but merely describe this is one kind of action (Spec. 8, l. 7). The Examiner applies Heinen to show that a user can send a message which is stored data at a remote unit and then reasons that because a variable is a named storage location capable of containing data that can be modified, storing data necessarily modifies a program variable. While we do not agree with the Examiner's reliance on the user messages taught by Heinen for the reasons discussed in this first section 1, we think Haban contains other teachings that reasonably suggest the claimed invention as discussed in the next section 2.

Appellants note that the Examiner admits that Haban does not explicitly disclose modifying a program variable in a second job. Appellants argue that the Examiner errs in the user command in Heinen, "DEPOSIT -- a message requesting that data forming part of the message is deposited in the memory of the specified job or process," because (1) depositing of information does not imply a modifying a program variable, and (2) the DEPOSIT command is a user command which teaches away from modifying a program variable in response to a condition. Br. 5.

First, the Examiner finds that a "variable" is defined as "a named storage location capable of containing data that can be modified during

program execution," *Microsoft Dictionary*, so storing data modifies a program variable. Ans. 6-7.

We agree with the definition. Since the program receiving the message must be able to utilize the data, the data must be represented by a program variable. However, we do not agree that Heinen would have suggested the claim difference for the reasons discussed below.

Second, the Examiner finds that Haban teaches (at § 1. Introduction and § 2.1 Primitive Events) examples of breakpoints wherein variables are consulted and manipulated in order to send a message and concludes:

> Those of ordinary skill in the art would have recognized that manipulation and monitoring of variables by a process that execute [sic] separately and do not share memory and thus relays related messages between one another, to illustrate modify of a program variable by at least (1) the definition of a variable or the (2) functions or instructions of the process to the variable (i.e. assigning variables function Table 1).

Ans. 7.

Appellants argue that the Examiner's reading is incorrect and that a correct reading of Haban is that Haban may monitor for when a program assigns a value to one of its own variables, which has nothing to do with modifying a program variable in a second job. Reply Br. 2-3. In addition, it is argued that assigning of a value by a program to one of its own variables has nothing to do with "manipulation and monitoring of variables by a process that execute [sic] separately." Reply Br. 3.

The Examiner's reasoning is unclear. We agree with Appellants that Haban teaches a program modifying the value of its own variables. The

Examiner has not explained how this teaches or suggests modifying a program variable in a second job. The Examiner mentions "messages" but does nothing to develop this line of reasoning.

Third, the Examiner states that the claim does not require that the modification be done "automatically" or by a "non-user action" and thus, Heinen's disclosure of a user command does not teach away because the claims are broad. Ans. 7.

Appellants argue that the claims require an inter-job mechanism that modifies a program variable in a second job "in response to" detecting a condition in the first job, and a user sending a DEPOSIT command is not done "in response to" detecting a global event. Reply Br. 3. It is argued that manually modifying a program variable in a second program, assuming that is what is done by Heinen, does not teach a mechanism for modifying a program variable "in response to" detecting a condition. Reply Br. 4.

We agree with Appellants that finding sending data to a second process does not explain how data is sent "in response to" detecting a condition.

The Examiner also refers to a statement in Appellants' Specification that the claims are not limited to a specific example. Ans. 7.

Appellants argue that this does not address the specific claim language of an inter-job mechanism that modifies a program variable in the second job "in response to" the mechanism detecting a condition in the first job. Reply Br. 4.

We agree with Appellants that the claim language is not without limit as the Examiner seems to infer.

Appellants argue that the rationale for combining Haban and Heinen is defective because the Examiner has provided only a very general motivation, "to provide means for debugging distributed processes," which does not provide motivation for the specifically claimed invention. Br. 6. It is argued that "[t]he motivation to combine must have some relationship to the combination in the claimed invention, and not be just some general benefit." Br. 7.

The Examiner provides additional reasoning. Ans. 9-10.

Appellants stand by the arguments in the Appeal Brief. Reply Br. 5.

We agree with Appellants that general statements of motivation, such as presented here, are not useful in explaining why one of ordinary skill in the art would have been motivated to make the combination. If a skilled person were given Haban and Heinen and told to combine them "to provide means for debugging distributed processes," as stated by the Examiner, we see no way that such guidance would produce the claimed subject matter. Motivation needs to be specific and particular. Although motivation need not be expressly found in a reference, any reasoning must be persuasive.

In summary, we are not persuaded that one of ordinary skill in the art would have been motivated by Heinen to modify Haban in the manner or for the reasons stated by the Examiner.

2.

Nevertheless, the Examiner has not addressed the best teachings of Haban, and we conclude that the claims would have been obvious to one ordinary skill in the art over Haban alone.

Haban describes that the message representing a primitive event consists of 1 byte data and 4 bytes for each additional parameters. These 5 bytes of message data have to be stored at the node processor receiving the message. Thus, Heinen is not necessary to show sending and storing data. Haban describes that "[w]ith the aid of a database gathered during unit compilation, the event processing software is able to interpret the incoming events and their parameters." P. 174, left col. Although Haban does not explicitly state that the message data and parameters modify program variables, one of ordinary skill in the art would have appreciated that if "the event processing software is able to interpret the incoming events and their parameters" (p. 174, left col.), the event data and parameters must be represented as program variables where the data and parameters modify the value of those program variables. In addition, Haban discloses that "users should be able to specify assertions such as, 'halt the system when variable x of process b minus variable y of process c gets lower than zero," p. 166, left col., which suggests to one of ordinary skill in the art that the values of the program variables x and y have to be sent from one process to another, at least from process b to process c, in order for the condition (x - y < 0) to be tested and trigger the action of halting the computer. For these reasons, Haban reasonably suggests to one of ordinary skill in the art "an inter-job

breakpoint mechanism that detects at least one condition in the first job and, in response thereto, modifies a program variable in the second job."

In sustaining a multiple reference rejection under 35 U.S.C. § 103(a), the Board may rely on one reference alone without designating it a new ground of rejection. *See In re Bush*, 296 F.2d 491, 496 (CCPA 1961); *In re Boyer*, 363 F.2d 455, 458 n.2 (CCPA 1966).

Conclusion

Haban suggests "an inter-job breakpoint mechanism that detects at least one condition in the first job and, in response thereto, modifies a program variable in the second job," as recited in representative claim 8. The rejection of claims 8, 17, and 28 is affirmed.

Claims 9, 18, and 29

Contentions

The Examiner finds that Haban does not teach outputting a debug message to a second job's output. The Examiner finds that Diec teaches outputting a debug message to a second job's output because it teaches "issuing a message to another software object to trigger generation of tracing data" (col. 2, ll. 1-5) and concludes that it would have been obvious to combine the teachings of Haban, Heinen, and Diec "to provide means for debugging distributed processes" (FOA 7) because all references deal with distributed processes.

Appellants argue that the Examiner errs because issuing a message to another software object as taught in Diec does not read on outputting a

debug message to the second job's output, i.e., the message to trigger generation of tracing data in Diec is not *in* the output from the job. Br. 8. It is argued that the Examiner's rejection has internal inconsistencies and the stated motivation is too general to be useful. Br. 9-10.

The Examiner explains in the Answer that Diec's "trace data" is considered to correspond to the claimed "debug message" and the trace data is relayed from software to component to software component until it is written to a "file output." Ans. 11. The Examiner also states that processes generally store an "output buffer" in memory and, therefore, Heinen's DEPOSIT command "could be used by those of ordinary skill in the art to 'deposit' data into such an output buffer." Ans. 11.

Appellants argue that the Examiner pulls the "output buffer" out of thin air and that "could be used" is not obviousness reasoning. Reply Br. 7.

Issue

Does the combination of Haban, Heinen, and Diec teach or suggest that "the inter-job breakpoint mechanism, in response to detecting the at least one condition in the first job, outputs a debug message to the second job's output" as recited in claim 9?

Dependent claims 18 and 29 contain corresponding limitations and thus, the rejection of claims 9, 18, and 29 stands or falls with representative claim 9.

Analysis

As noted by the Examiner, the limitation that "the inter-job breakpoint mechanism, in response to detecting the at least one condition in the first job, outputs a debug message to the second job's output" "does not indicate specific steps which [are] taken to output [a] 'debug message'; and further does not specify what form the debug message must take or where the message comes from." Ans. 10. The support in the Specification for the limitation is that "[e]xamples of actions 128 within the scope of the preferred embodiments include . . . the modifying of a variable or other property on the second job, and the outputting of a debug message to the second job's output." Spec. 8, ll. 5-8. No specific details are disclosed about outputting the debugging message to the second job's output, such as the nature of the debugging message or how the debug message is output. Accordingly, the limitation is broad. We note that outputting a debug message in claim 9 is in addition to modifying a program variable in parent claim 8.

Although the claim is broad and we agree with the Examiner that Diec teaches issuing a message to another software object, which message could be considered a "debug message," we agree with Appellants that this does not show how the message is output to the second job's output. The Examiner finds that issuing a message and channeling an entry into a log file in Diec teaches that the message is passed through multiple software components. However, the cited portions of Diec only disclose sending a message which triggers generation of tracing data (i.e., debug data), or recording a entry, not outputting a debug message to a second object's

output. In addition, the Examiner's stated motivation for the combination is not specific. The fact that all the references related to distributed systems does not point to the particular claimed invention.

Nevertheless, it appears that claim 9, as broadly recited, would have been obvious over Haban alone. A primitive event occurring in a process in Haban is detected condition. P. 166, right col. A "global event" is composed of one or more primitive events. *Id.* The notification that a primitive events has occurred is sent by messages from one process to another as indicated by the arrows in the figures, where these messages are broadly considered "debug messages." Therefore, if event Ei happens in process Pi (a first job) and a message is sent to process Pj (a second job) and then an event Ej happens in process Pj and a message is sent to process Pk (a third job), and then an event Ek happens in Pk, as illustrated in Figure 13 of Haban, the message that event Ei has occurred is a debug message output through the output of process Pj. Since the nature of the debug message and how it is output is not claimed or described, this meets the claim terms.

Conclusion

Haban suggests that "the inter-job breakpoint mechanism, in response to detecting the at least one condition in the first job, outputs a debug message to the second job's output" as recited in claim 9. Accordingly, the rejection of claims 9, 18, and 29 is affirmed.

CONCLUSION

The rejections of claims 8, 9, 17, 18, 28, and 29 under 35 U.S.C. § 103(a) are affirmed.

Requests for extensions of time are governed by 37 C.F.R. § 1.136(b). *See* 37 C.F.R. § 41.50(f).

<u>AFFIRMED</u>

peb

MARTIN & ASSOCIATES, LLC
P.O. BOX 548
CARTHAGE, MO 64836-0548